METHOD AND SYSTEM FOR APPLYING INPUT MODE BIAS

TECHNICAL FIELD

The invention generally relates to software program modules, and even more particularly, relates to using grammars to apply input mode bias to portions of an electronic document, including fields, and using the same grammars to apply semantic categories to strings in the document.

BACKGROUND

Computer software is becoming more and more accessible to receiving inputs from many different input methods such as through speech recognition, handwriting recognition, and East Asian input method editors, just to name a few. However, these stochastic input methods sometimes are less than reliable. For example, if speech recognition software is unaware of the particular type of input required in a particular field, then it may be difficult for it to understand what the user is trying to say. For

example, it may be difficult to enter a phone number due to the speech recognition software being unaware of the particular type of input required.

Solutions have been developed that constrain input so that authors of text entry controls may set a property on a field so that an input method editor will recognize the bit and adjust the language model that the input method editor uses. Constraining input of a field, constraining input of part or all of a document, etc. may be referred to as mode bias. However, there are some drawbacks to these solutions. These solutions are limited to a fixed list of categories that may be indicated by the bit. Thus, there is an inflexible hard-coded list of ways to constrain input. A developer must select from the fixed list of categories and is not allowed to define new mode bias settings that may be useful.

Another problem with the current solution of using a fixed list of categories is that the response to these categories is not consistent between different input methods. A user expects different input methods to react similarly for the same input field. However, prior art solutions allow input methods to define their own response to particular categories of text entry controls. For example, a speech recognizer and a handwriting recognizer may handle a phone number entirely differently.

Thus, there is a need for a method for providing mode bias that is flexible in allowing developers to define what forms of input are acceptable for a particular field. There is still a further need for a method for providing mode bias that is consistent across different input methods.

SUMMARY OF THE INVENTION

The present invention meets the above-described needs by providing a method and system for applying input mode bias to input fields (and other regions) of an electronic document. A schema is applied to an input field of an electronic document and the schema is associated with a grammar that defines the set of acceptable strings that may be input into the input field. When text is input into the input field, the schema is determined and the associated grammar is determined. The grammar is then sent to the input method. Based on the grammar, the input method may determine what text the user is trying to enter. The grammar may also be used to apply semantic categories to text

input into a document. After the text is entered it is compared to a grammar. If the entered text matches the form of the grammar, then the schema is applied to the text as a semantic category.

Typically, the schema names are stored in a schema database and the grammars are stored in a grammar database. The schema names may also be stored with other properties such as locale and language and a pointer to the associated grammar. Thus, when a user places the insertion point in an input field with mode bias applied, then the schema name and other properties are sent to the schema database for a look-up of the appropriate grammar. Based on the schema name (and other properties), the grammar (or a pointer to the grammar) is sent to the input method. The input method will then use the grammar to determine what the user is trying to enter into the input field. The input method may also provide suggestions or error messages if the user attempts to enter text that does not conform to the grammar.

Because the schema database and grammar database are outside of the control of the input methods, the present invention improves upon the prior art problem that input methods would handle mode bias differently. Using the present invention, each input method receives the same grammar and should handle it similarly whether the input method is a speech recognition engine, handwriting recognition engine, etc. The present invention is also flexible because new schemas and grammars may be defined and implemented by system administrators to provide custom mode bias that may be needed within an organization. In addition to providing mode bias, the grammars may be used to recognize semantic categories in a document.

That the invention improves over the drawbacks of the prior art and accomplishes the advantages described above will become apparent from the following detailed description of the exemplary embodiments and the appended drawings and claims.

BRIEF DESCRIPTION OF THE FIGURES

Fig. 1 is a block diagram of a computer that provides the exemplary operating environment for the present invention.

Fig. 2 is a block diagram illustrating an architecture for implementing mode bias

in accordance with an embodiment of the present invention.

Fig. 3 is a flowchart illustrating a method for applying mode bias to text in accordance with an embodiment of the present invention.

Fig. 4 is a block diagram illustrating an exemplary architecture for implementing semantic categories.

Fig. 5 is a flow chart illustrating a method for semantically labeling strings during creation of an electronic document.

Fig. 6 is an illustration of a display of a semantic category and its associated dropdown menu.

Fig. 7 is a flowchart illustrating a method for semantic category recognition using grammars in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

The present invention is directed toward a method and system for applying input mode bias. A schema is applied to an input field of an electronic document. The schema is the name of the type of input that should be input into the input field. As an example, the schema may be a telephone number schema. Associated with the schema is a grammar that defines the set of acceptable strings that may be input into the input field. For example, the grammar associated with the telephone number schema may define the input as being of the form (XXX) XXX-XXXX, where X represents a digit from 0-9.

Having briefly described an embodiment of the present invention, an exemplary operating environment for the present invention is described below.

Exemplary Operating Environment

Referring now to the drawings, in which like numerals represent like elements throughout the several figures, aspects of the present invention and the exemplary operating environment will be described.

Fig. 1 and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. While the invention will be described in the general context of an application program that runs on an operating system in conjunction with a personal

computer, those skilled in the art will recognize that the invention also may be implemented in combination with other program modules. Generally, program modules include routines, programs, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to Fig. 1, an exemplary system for implementing the invention includes a conventional personal computer **20**, including a processing unit **21**, a system memory **22**, and a system bus **23** that couples the system memory to the processing unit **21**. The system memory **22** includes read only memory (ROM) **24** and random access memory (RAM) **25**. A basic input/output system **26** (BIOS), containing the basic routines that help to transfer information between elements within the personal computer **20**, such as during start-up, is stored in ROM **24**. The personal computer **20** further includes a hard disk drive **27**, a magnetic disk drive **28**, e.g., to read from or write to a removable disk **29**, and an optical disk drive **30**, e.g., for reading a CD-ROM disk **31** or to read from or write to other optical media. The hard disk drive **27**, magnetic disk drive **28**, and optical disk drive **30** are connected to the system bus **23** by a hard disk drive interface **32**, a magnetic disk drive interface **33**, and an optical drive interface **34**, respectively. The drives and their associated computer-readable media provide nonvolatile storage for the personal computer **20**. Although the description of computer-readable media above refers to a hard disk, a removable magnetic disk and a CD-ROM disk, it should be appreciated by those skilled in the art that other types of media which are readable by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, and the like, may also be used in the exemplary operating

environment.

A number of program modules may be stored in the drives and RAM **25**, including an operating system **35**, one or more application programs **36**, a word processor program module **37** (or other type of program module), program data **38**, and other program modules (not shown).

A user may enter commands and information into the personal computer **20** through a keyboard **40** and pointing device, such as a mouse **42**. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, stylus, or the like. These and other input devices are often connected to the processing unit **21** through a serial port interface **46** that is coupled to the system bus, but may be connected by other interfaces, such as a game port or a universal serial bus (USB). A monitor **47** or other type of display device is also connected to the system bus **23** via an interface, such as a video adapter **48**. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers or printers.

The personal computer **20** may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer **49**. The remote computer **49** may be a server, a router, a peer device or other common network node, and typically includes many or all of the elements described relative to the personal computer **20**, although only a memory storage device **50** has been illustrated in Figure 1. The logical connections depicted in Figure 1 include a local area network (LAN) **51** and a wide area network (WAN) **52**. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the personal computer **20** is connected to the LAN **51** through a network interface **53**. When used in a WAN networking environment, the personal computer **20** typically includes a modem **54** or other means for establishing communications over the WAN **52**, such as the Internet. The modem **54**, which may be internal or external, is connected to the system bus **23** via the serial port interface **46**. In a networked environment, program modules depicted relative to the personal computer **20**, or portions thereof, may be stored in the remote

memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Applying Input Mode Bias

Stochastic input methods are being used more frequently today than ever before. For some time stochastic input mechanisms have been used in East Asia in the form of Input Method Editors (IMEs). Recently, speech and handwriting input mechanisms have become more common. Unlike typing (outside of East Asia) where the user is able to easily type whatever he chooses, stochastic input methods by their nature presuppose that the user is trying to enter more common words and phrases. This presupposition typically works well when the user intends to enter common words and phrases. However, it can make it difficult to enter certain types of less common text. This can be particularly frustrating when inputting text into a form where there are constraints placed on what sort of information should be entered and the input mechanism doesn't adhere to these constraints. For example, it makes little sense to enter anything other than digits into a phone number field in contact management software, but speech and handwriting recognizers are rarely cognizant of this sort of constraint. In one embodiment, the present invention solves this problem with a mechanism for controlling mode bias of input controls, or input fields.

In one embodiment, the present invention may be used to define an XML schema associated with an input control. The schema may be associated with a grammar. A grammar is a definition of appropriate input (e.g. acceptable strings) for the input control. In one embodiment, the grammar may be a context free grammar, statistical language model, a list of possible inputs, etc.

The grammar is used to constrain the user's input so that the user completes an input field properly. For example, an input control may specify that it accepts part numbers by attaching <schemas-microsoft-com#partnumber> to the control. The schema microsoft-com#partnumber may be associated with a grammar that specifies that this specific sort of part number comprises a digit from 0 to 9 followed by a letter from a-z.

In the example described above, the grammar is used to constrain the user's input to match the form of a part number. If the user enters something other than a part number, an error message or a suggestion may be presented to the user.

In other embodiments, the grammar may be used by different input methods to correctly identify the user's input. For example, a user of a speech recognition engine may say "four". Using the present invention, the speech recognition input method editor may determine whether the user meant "four", or "for". If the user is entering text into an input field that has a phone number schema attached to it and is defined by a grammar that accepts numbers, then it is likely that the user meant the word "four". Thus, the speech recognition input method editor will recognize that it should enter the numeral "4" into the input field.

Fig. 2 illustrates an architecture **500** for implementing mode bias in accordance with an embodiment of the present invention. An application, such as application **37**, is connected to a text services framework **505**. The text services framework allows input from stochastic input sources. The text services framework is an application programming interface (API) set that allows input sources to communicate with applications. The text services framework is the successor to both the IMM32 (East Asian input method API for the WINDOWS operating system) and AIMM (active input method manager). The text services framework provides numerous advantages over IMM32 and AIMM such as wider access to context, the ability to associate arbitrary properties with text, etc. Although the text services framework is used in one embodiment of the invention, it should be understood that in other embodiments the invention may include other tools for connecting the application to different input methods.

The application **37** may optionally support semantic categories as described below. Input methods, such as a speech recognition engine **525**, a handwriting recognition engine **530**, input method editors **535** and other stochastic input mechanisms **540** (e.g. sign language recognition) allow the user to enter text into a document of application **37** via the text services framework **505**. Other input mechanisms such as

gesture-based recognition engines that determine input based on movements or gestures of the user may be used. Input methods such as the keypad of a cellphone or personal digital assistant (PDA) may also be used.

The text services framework **505** and the different input methods are connected to a schema registry. In a preferred embodiment, the schema registry comprises a schema registry manager **510** connected to a grammar database **515** and a schema database **520**.

In a preferred embodiment, the schema database comprises a plurality of XML schema used to control mode bias. XML schema are preferably used because they're general and flexible and are already being used for semantic category information (semantic categories are described below). XML schema may also be customized as needed to extend the present invention to new types of data (for example, part numbers within a large organization that conform to a standardized syntax). Although new schema will need to be defined for some inputs, standard schema used in semantic categories may be reused as schema in the present invention.

In order to maximize the ability to reuse grammars for standard broadly applicable types of input controls (such as phone numbers) and for types of input controls that are standard within an organization (such as part numbers), in a preferred embodiment of the present invention the schema either refer to grammars directly or map to them via the schema registry that associates each schema with a particular grammar. For example, the schema names may be referred to in the schema database **520** along with an associated grammar(s) for the schema name. The grammars themselves may be stored in the grammar database **515**. One advantage of this indirect mapping approach is that it allows for additional abstraction across, for example, language or region. For example, a field may specify that it accepts phone numbers without needing to build the grammars for different phone formats for each user locale. The schema registry manager **510** controls both the schema database **520** and the grammar database **515**. Of course, other architectures will be apparent to those skilled in the art and may be implemented without departing from the present invention.

In other embodiments, the XML schema could directly encode the grammar, but that would make sharing of grammars more difficult. For example, to directly encode the grammar, the XML schema could include the grammar in the form of XML tags that described the list of or form of acceptable input (rather than pointing to a grammar file such as numbers.xml).

Having a schema registry system is beneficial because it easily allows independent software vendors and sophisticated users to modify existing schema or define their own schema and have them used within applications by input methods and for semantic category recognition immediately—that is, without waiting for the next version of the application or input method that is savvy about this new schema. For example, a developer within an organization may notice that users are having a difficult time entering certain types of input when using stochastic input methods (such as when entering part numbers that are specific to the organization). So, the developer may create a new schema to describe part numbers and a grammar to recognize these part numbers. The developer may then attach this schema to form fields in their application and their web pages. The end user may do this as well by annotating a field in a form he created with this new schema.

The schema registry may include entries like those in the first two columns in Table 1 below. The third column explains what the grammar is meant to do. In some cases the grammar may be built dynamically based on the contents of the user's computer, exchange server, etc. Thus, one way that the present invention constrains an input method to adhere to a particular pattern is to specify a grammar that the input method should use. In other words, the invention specifies a regular expression, context free grammar (CFG) or statistical language model that the input method should use. It should be understood that Table 1 is illustrative only and does not limit the number or types of schemas that may be used in the present invention.

Schema name (namespace#tag)	Grammar	Interpretation of grammar
Schemas-microsoft-com-modes#date	A simple regular expression: (January|February|March|...) digit digit	Permits dates to be entered in Month Date format (e.g. January 12)
Schemas-microsoft-com-modes#personname	A context free grammar (CFG) built from the union of the contacts list, the global address book, and lists of first and last names	Almost any name
Schemas-microsoft-com-modes#contact	A CFG built exclusively from the contents of the contacts list	Any name in the contacts folder
Schemas-microsoft-com-modes#global_address_list	A CFG built exclusively from the contents of the global address list (GAL)	Employees of the company the user works in
Schemas-microsoft-com-modes#number	Digit+	An arbitrary sequence of digits
Schemas-microsoft-com-modes#legalText	A statistical language model (SLM) for legal text entry	Allows entry of legal text

Table 1

Thus, in one embodiment, the schema registry comprises an XML schema name (comprising a namespace and a tag) and a grammar associated with the XML schema name. The schema registry may also comprise other defining properties that are necessary to determine which grammar to use. A language setting and a locale setting may be used in the schema registry to further define which grammar to use. For example, an address schema may be associated with several different grammars. The grammar that should be sent to the input method may be determined by looking at the language and locale of the computer (or operating system, application, etc.) and then performing a database look-up in the schema registry to locate the appropriate grammar. The schema

registry may also include a semantic category setting to indicate whether a particular schema name and grammars should be used for semantic category recognition (as described further below).

The schema registry may be hierarchical. In other words, several hierarchical schemas may be associated with an input control. For example, searching for particular names, such as those in the contacts folder, may be helpful in some cases. However, ruling out other names may be too extreme. As a result, an input field might best be described as expecting input biased towards names in the contacts folder but permitting other names to be entered. Organizing schemas hierarchically addresses this. For example, the Personname schema may be at the top of the name hierarchy while a contacts schema and corporate employee schema may be hierarchically below the Personname schema. A large organization may define the corporate employee schema to make it easier to enter employee names into form fields. Thus, in this example, when a field calls for a Personname schema, the contacts schema and corporate employee schema may be searched before searching for other person's names. In other words, the schemas may be hierarchical in that each schema may return a grammar to the input method and the input method may prefer the most specific schema or otherwise combine the schemas.

There are also cases where a user might try to fill out a form that asked for a schema that the user's machine doesn't know about. For these cases, an embodiment of the present invention may specify an ordered sequence of preferred schemas. The best schema would then be used if it was in the registry or if it could be downloaded in time. However, if it was not available the input method may fall back on the second favorite schema, and so on. For example, a user may try to fill out a form which requests the type Microsoft part number, but the user might not have that type of schema (and/or grammar associated with the schema) on their machine. Much better than allowing arbitrary input then would be to fall back on sequences of letters and digits rather than arbitrary input, which might be predisposed to favor words (as is the case for most speech engines).

Having briefly described an architecture for implementing the present invention, a flowchart illustrating a method **600** for applying mode bias to text in accordance with an embodiment of the present invention will be described in reference to Fig. 3. It should be understood that the method **600** occurs in an application after the author of a template (or form), or the creator of an application, attaches mode bias to a region of text or a field. A tool such as the "VISUAL STUDIO" application or another similar tool may be provisioned in the future to allow an author to apply such mode bias.

At step **605**, the insertion point (IP) is placed into a region of an electronic document with mode bias attached (such as an XML schema attached). At step **610**, the input method the user is using retrieves the ranked list of mode biases (schemas) (e.g. best schema=Microsoft employee, 2nd best=person name) from the application **37** through the operating system. At step **615**, the input method (or operating system) calls into the schema registry and retrieves the grammars associated with the highest ranked schema. Alternatively, the input method (or operating system) calls into the schema registry and retrieves all of the grammars associated with all of the schemas associated with the insertion point region.

At step **620**, the user enters text using an input method and the input method recognizes text using the grammars from the registry. Text is inserted into the application as the input methods use the grammars to determine what the user is inputting. The schema, or series of schema, may also be inserted into the document along with the text. When the document is saved, the text is saved, and in some cases so is the schema applied to that text.

It should be understood from the foregoing description that if a document managed by an application has mode bias set on a particular field (or region within a document) that information would be conveyed to the input method. In a preferred embodiment, the information is conveyed from the application through the text services framework. However, it should be understood that other methods of conveying this mode bias information to the input method may be used without departing from the present invention. After receiving the information, the input method would then interface with

the schema registry manager in order to identify an appropriate grammar in the grammar database to use. Typically, the input method requests that the schema registry manager return a grammar for the XML tag used to apply mode bias in the document. The schema registry manager would then pass this grammar (or more likely a pointer to it) to the input method for use in determining the user's input.

Thus, in one embodiment, the invention provides a mechanism for using schema (XML tags) to apply mode bias to inputs. The schema tags are used by a stochastic input method to retrieve grammars for use during user input. The grammars define what may be input into certain fields. For example, a form may specify that it accepts part numbers by attaching <schemas-microsoft-com#partnumber> to the control. That schema may be associated with a grammar that specifies that this specific sort of part number comprises a digit from 0 to 9 follow by a letter from a-z. That grammar would constrain the user's input to match that form of part number. The input method would then be able to understand what the user is attempting to input and may provide hints, examples or error messages if the user attempts to enter input that does not match the form defined by the grammar. The same grammar could also be used to recognize part numbers and apply the <schemas-microsoft-com#partnumber> semantic category to free text the user edits (as will be described below). It should be understood that the schema <schemas-microsoft-com#partnumber> is only one example. Schemas may be widely defined and the above example is not meant to limit the schemas in any way.

In one embodiment of the present invention, the schema registry manager itself may be responsible for generating grammars, e.g., the schema may point to code for generating the grammar. This embodiment may be particularly useful for grammars that depend on dynamic databases, such as company e-mail addresses. Rather than having to update each user's computer with revisions to the e-mail addresses, an administrator may set the schema to point to code for generating the grammar so that the latest e-mail addresses may be retrieved.

An application may provide properties along with schema. The properties may be used to limit or further define a proper input for the input field. For example, the schema

may be a telephone number and the properties are that it is not a telephone number beginning with the (555) area code. The schema registry manager may interpret the properties to modify the grammar or modify which selections of grammars are sent to the input method.

The use of XML schema and grammars has been described above with regard to applying mode bias to input fields. After XML schema and grammars have been implemented as described above, they may be used to drive recognition of semantic categories because the same grammars used during input with a stochastic input method can be used after text is entered to identify semantic categories.

Semantic Category Recognition Using Grammars

Some of the grammars described above in relation to mode bias may be used for recognition of semantic categories. In a preferred embodiment, a semantic category comprises a type label, a download URL and metadata that is associated with a string that is recognized in a document. A string is a data structure composed of a sequence of characters usually representing human-readable text. When a user enters a string into a document, semantic categories may be labeled for the string and specific actions may be presented to the user in association with the semantic categories. For example, when a user enters the title of a book into a document, it may be labeled as a book and certain actions such as "Buy the book" may be presented to the user based on the labeling.

Before proceeding with a description of using grammars to recognize semantic categories, a more detailed description of semantic categories is provided below. Semantic categories are discussed further in U.S. Patent Application Serial No. 09/588,411, entitled "METHOD AND SYSTEM FOR SEMANTICALLY LABELING STRINGS AND PROVIDING ACTIONS BASED ON SEMANTICALLY LABELED STRINGS", filed June 6, 2000, which is commonly assigned and incorporated by reference herein.

To implement semantic categories, strings are recognized and annotated, or labeled, with a type label. After the strings are annotated with a type label, application program modules may use the type label and other metadata to provide users with a

choice of actions. If the user's computer does not have any actions associated with that type label, the user may be provided with the option to use a download Uniform Resource Locator (URL) and download action plug-ins for that type label.

Fig. 4 is a block diagram illustrating an exemplary architecture **200** for use in conjunction with labeling semantic categories in a document. The architecture includes an application program module **205**, such as program module **37** (Fig. 1). The application program module **205** is able to communicate with a recognizer dynamic-link library **210** (hereinafter recognizer DLL) and an action dynamic-link library **215** (hereinafter action DLL) as a user is creating, editing, viewing, etc. an electronic document. The recognizer DLL **210** controls a number of recognizer plug-ins **220**. The action DLL **215** controls a number of action plug-ins **225**. The action DLL also controls a type-action database **230**.

The recognizer DLL **210** handles the distribution of strings from the electronic document running on the application program module **205** to the individual recognizer plug-ins **220**. The recognizer plug-ins **220** recognize particular strings in an electronic document, such as a word processing document, a spreadsheet document, a web page, etc. The recognizer plug-ins **220** may be packaged with the application program module **205** or they may be written by third parties to recognize particular strings that are of interest. Typically, the recognizer DLL **210** passes strings to the recognizer plug-ins **220** in one paragraph or cell value increments. It should also be understood that, in a preferred embodiment, the action DLL and recognizer DLL are merged into a single DLL.

As part of recognizing certain strings as including semantic information, the recognizer plug-ins **220** determine which strings are to be labeled and how they are to be labeled. After receiving these results from the various recognizer plug-ins **220**, the recognizer DLL **210** sends semantic categories to the application program module. In a preferred embodiment, a semantic category comprises the recognized string, a type label, and a download URL. A semantic category may also comprise metadata. The recognizer plug-ins **220** each run separately and the recognizer DLL **210** is responsible for handling

the asynchronicity that results from different recognizer plug-ins returning results with different delays.

After a string is labeled by a recognizer plug-in **220** and a semantic category is sent to the application program module **205**, the user of the application program module **205** will be able to execute actions that are associated with the type label of the semantic category. The action DLL **215** manages the action plug-ins **225** that are run to execute the actions. As with the recognizer plug-ins **220**, the action plug-ins **225** may be packaged with the application program module **205** or written by third parties to perform particular actions that are of interest to the third party. The action plug-ins provide possible actions to be presented to the user based upon the type label associated with the string. The action DLL **215** determines what type label the semantic category includes and cross-references the type label in the type-action database **230** with a list of actions to determine what actions to present to the user. It should be understood that, in a preferred embodiment, the type-action database is not used. Instead, the list of actions is dynamically generated for each type by looking in the registry to determine which actions are installed and then querying the action DLLs to determine which types they apply to.

After the user chooses an action, the action DLL **215** manages the appropriate action plug-ins **225** and passes the necessary information between the action plug-ins and the application program module **205** so that the action plug-in may execute the desired action. Typically, the application program module sends the action DLL an automation request to invoke the action the user has selected.

As described above, the combination of the recognized string, type label, metadata and download URL is referred to herein as a semantic category. The type label is a semantic information label. The semantic category may also comprise metadata, which are hidden properties of the semantic category. An example of a semantic category may clarify the definition. Suppose a user enters the text "Gone With the Wind" into an electronic document. The string "Gone With the Wind" may be identified as a semantic category of type label "Book Title" and of type label "Movie Title". In addition, metadata such as the ISBN number may be returned by the recognizer plug-in to

the application program module as part of the semantic category. A download URL may be provided with the type labels "Book Title" and "Movie Title" in case the user's machine has not stored action plug-ins for these type labels. For example, an action for the type label "Book Title" may be "Buy this Book" from an online retailer. If the user does not have the action plug-in DLL **225** corresponding to "Buy this book", then the download URL may be used to navigate the user's web browser to an appropriate website to download this action plug-in. In other implementations of the invention, multiple download URLs may be provided for a single type label.

Having described an exemplary architecture, an exemplary method **300** for semantically labeling strings during document creation will be described below in reference to Figs. 4 and 5.

Fig. 5 is a flow chart illustrating a method **300** for semantically labeling strings during creation of an electronic document. Those skilled in the art will appreciate that this is a computer-implemented process that is carried out by the computer in response to input from the user and instructions provided by a program module.

Referring to Fig. 5, the method **300** begins at start step **305** and proceeds to step **310** when a user opens an electronic document in application program module **205**. In a preferred embodiment, the electronic document is a word processing document or a spreadsheet document. However, the method **300** is not limited to either of these specific types of electronic documents.

At step **310**, the application program module **205** receives a new string, such as when the user enters a new paragraph into the electronic document or edits a previously entered paragraph. The method **300** then proceeds to step **315**.

At step **315**, the paragraph containing the new string is passed from the application program module **205** to the recognizer DLL **210**. The recognizer DLL is responsible for communicating with the application program module, managing the jobs that need to be performed by the recognizer plug-ins, receiving results from the recognizer plug-ins and sending semantic category information to the application program module. At boot time, the recognizer DLL communicates with its recognizer

plug-ins to determine what languages it supports, what types it can apply, etc. It should be understood that, in a preferred embodiment, a paragraph is passed to the recognizer DLL at step **315**. However, in alternative embodiments, a sentence, the contents of a spreadsheet cell, a section of the document, the entire document, etc. may be passed to the recognizer DLL. In other words, the present invention is not limited to simply passing a paragraph to the recognizer DLL. The method **300** then proceeds to step **320**.

Still referring to step **315**, the application program module **205** typically sends one paragraph at a time to the recognizer DLL. In addition, in a preferred embodiment, a grammar checker program module sends all semantic categories (without type labels) to the recognizer DLL that have been identified by the grammar checker program module. Passing these semantic categories (without type labels) to the recognizer DLL is important because doing so saves each recognizer plug-in from needing to decide whether something is a capitalized string interspersed with function words (a task that would require writing a number of regular expressions: Cap Cap Unc Cap; Cap Unc Cap; etc.). If a label is applied by a recognizer plug-in to a string the grammar checker program module labeled, the grammar checker label will then be removed.

At step **320**, during idle time, the paragraph (and information from the grammar checker program module) is passed to the recognizer plug-ins. The method then proceeds to step **325**.

It should be understood that, in a preferred embodiment, the recognizer DLL **210** maintains a job queue. If before the recognizer DLL **210** sends the paragraph to the recognizer plug-ins **220** the user edits the paragraph, then the job containing the edited paragraph is deleted and is not sent to the recognizer plug-ins. Then, a new job enters the queue at step **315** after the edited paragraph is received at step **310**. This job deletion is necessary to prevent the recognizer plug-ins from performing unnecessary work on a paragraph that has been edited.

At step **325**, the recognizer plug-ins are executed on the paragraph to recognize keywords or perform other actions defined by the recognizer plug-in. As part of executing the recognizer plug-in, the paragraph may be broken into sentences by the

recognizer plug-in. However, each recognizer plug-in is responsible for its own sentence-breaking. After the keywords are found at step **325**, then the method proceeds to step **330**.

At step **330**, the results from each of the recognizer plug-ins are received by the recognizer DLL. The method then proceeds to decision step **335**.

At decision step **335**, it is determined whether the paragraph that has been reviewed by the recognizer plug-ins has been edited after the paragraph was sent to the recognizer DLL. If so, then the method **300** returns to step **315** and the edited paragraph is received by the recognizer DLL from the application program module. If not, then the method proceeds to step **340**.

At step **340**, the results from the recognizer plug-ins are compiled into semantic categories by the recognizer DLL and the semantic categories are sent to the application program module. At step **345**, the application program module displays the semantic categories to the user in the electronic document. The method **300** then ends at step **399**.

As should be understood from the description above, the architecture for recognizing semantic categories permits third parties to develop recognizer plug-ins to identify strings of one or more particular types. The recognizer plug-ins communicate with the application program module and receive a string from the application program module. The recognizer plug-ins may apply recognition algorithms to the string and communicate the identity of recognized strings back to the application program module.

After a string is labeled with a particular type label, the user will be able to execute action plug-ins that pertain to that type label. The action plug-ins preferably are COM objects that are executed via communication between the application program module and the action DLL. Parameters necessary to execute the action (the HTML of the string labeled as being of a particular type, the HTML of the string representing the current selection) will be passed from the application program module to the action DLL and, in turn, passed to the action plug-in.

An architecture for identifying and executing a set of actions associated with a semantic category may also be provided. This architecture comprises actions that apply

to a particular type label (e.g. an action for book titles may be "Buy this book from shop.Microsoft.com") and executing those actions when the user so desires. An action is a user-initiated function applied to a typed string. For example, adding a name to the contacts folder is one action possible for a type label "Person name".

Different actions may be assigned to different type labels and these type label-action assignments may be stored in the type-action database **230**. Table 2 below illustrates some possible type label-action pairings.

Type Labels	Actions
Person name	Show contact info Add to contacts E-mail Insert address into document Send instant message to
Date	Show calendar for that day New task with that due date Schedule meeting that day
Place	Display EXPEDIA map Add to contacts
Address	Add to contacts
Phone number	Add to contacts
E-mail	Add to contacts
Date	Schedule a meeting
Task	Schedule a task
Meeting	Schedule a meeting

Table 2

For each type label, the type-action database **230** may store a download URL specified by the creator of the type label that users who do not have action-plug-ins or

recognizer plug-ins for that semantic category type can go to in order to get action plug-ins and/or recognizer plug-ins. For example, the download URL for the type label "Book Title" might be microsoft.com/semanticcategories.asp. Once at that web page, a user may be offered downloads of various action plug-ins and recognizer plug-ins. There may also be an option on the user interface to navigate to the download URL so that recipients of documents with semantic categories can easily get the action plug-ins for those semantic categories.

Semantic categories may be saved as a unique namespace plus a tag name. A namespace is an XML construct for uniquely identifying a group of XML tags that belong to a logical category. Thus, every semantic category is uniquely identified by its nametag (e.g., "streetname") in addition to its namespace (e.g., "schemas-microsoft-com:outlook:contact")

Although the method **300** described above is one method for identifying semantic categories, there may be other mechanisms for identifying semantic categories. One mechanism is a grammar checker program module (not shown) connected to word processor program module **37**. Another mechanism is receiving a semantic category from another electronic document. For example, when text containing a semantic category is copied from one electronic document and pasted into another electronic document of the word processor program module **37**, the information identifying the semantic category is preserved and copied along with the copied text.

Referring now to Fig. 6, an illustration of a display of a semantic category **400** and its associated dropdown menu **405** will be described. It should be understood that Fig. 6 is an illustration of a semantic category **400** and dropdown menu **405** as displayed to a user by the application program module **205**.

The string **410** associated with semantic category **400** is the string "Bob Smith". As shown in Fig. 6, the string **410** of a semantic category **400** may be identified to the user by brackets **415**. Of course, many other devices such as coloring, underlining, icons, etc. may be used to indicate to the user that a particular string is a semantic category.

In a preferred embodiment, when the user hovers a cursor over the string **410** or

places the insertion point within string **410**, then dropdown menu **405** is displayed to the user. The dropdown menu may display a list of actions associated with a semantic category. The dropdown menu may appear above and to the left of the semantic category string.

Typically, the first line of the dropdown menu indicates which string is the semantic category string (Bob Smith in Fig. 6) and what type the semantic category is (Person name in Fig. 6). Listed below the first line are actions **420** available for the semantic category type, such as "Send mail to…", "Insert Address", and "Display contact information…".

The first item on the drop down menu below the separator line is "Check for new actions…" **425**. "Check for new actions…" **425** will appear only for semantic categories whose download URL is available to the application program module. If selected, "Check for new actions…" **425** uses the semantic category download URL to navigate the user's web browser to the homepage for the semantic category type applied to the string. For example, suppose new actions have been defined for the semantic category type "person name". If so, then new actions will be downloaded to the user's computer after selecting "Check for new actions…" **425**. "Check for new actions…" **425** will be grayed out if a download URL is unavailable for the semantic category.

If selected, the "Remove this semantic category" item **430** deletes the semantic category label from the string. If selected, the "Semantic categories" item **435** navigates the user to the semantic categories tab of the autocorrect dialog.

It should be understood that the application program module sends a request to the action DLL to determine which actions are shown with each semantic category type.

As described above with reference to Fig. 6, the application program module may include the option to display an in-document user interface to indicate the location of semantic categories. This in-document user interface may use a colored indication to indicate the location of a semantic category, such as the brackets **415** in Fig. 6. The in-document user interface will also be able to show nesting of semantic categories. For example, if Michael Jordan is labeled as a semantic category with type label "Person

Name", Michael is a semantic category with type label "First Name" and Jordan is a semantic category with type label "Last Name", the document may look like this with the brackets indicating semantic categories:

[[Michael][Jordan]]

Of course, the in-document user interface may be any sort of indication. For example, in the "EXCEL" spreadsheet application program, the interface comprises a triangle in the lower right hand portion of a cell to indicate that one or more semantic categories are present in the cell.

As described above, the semantic category may also include metadata returned by the recognizer plug-ins. For example, a recognizer plug-in that recognizes the titles of books may return as metadata an ISDN book number when it recognizes the title of a book. The ISDN book number metadata may then be used to provide actions. Metadata may also be used to disambiguate for actions and searches. For example, suppose a recognizer DLL is linked to a corporate employee database to recognize names. When the recognizer DLL recognizes "Bob Smith", it may store "employeeID=12345" as metadata in the background. Then, when an action is fired, the text in question will be known to reference Bob Smith, employee no. 12345 rather than Bob Smith, employee no. 45678. Also, the metadata may allow searches to be performed independent of the actual text in a document. So, a search may be conducted on "Robert Smith" by looking for employee 12345 in the employee databases and by performing a search on the metadata for employee number 12345 to find documents with "Bob Smith" in them. There are also numerous other functions for metadata. For instance, DHTML could be inserted so special tricks may be performed within a web browser. Additionally, data used by other actions may be inserted such as someone's e-mail address that could be used by the send-mail-to action, a normalized version of the date could be stored to easily interact with a personal information manager, etc.

Having described semantic categories in detail above, the description below focuses on providing semantic category recognition using grammars and XML schemas that are also used to apply mode bias. It should be understood that, in addition to

applying mode bias as described above, the application **37** (assuming it supports semantic categories) would also be passing text to the set of recognizer plug-ins via the infrastructure described in Fig. 4. One or more of the recognizer plug-ins may be connected to the grammar database **515** and schema database **520**. These recognizer plug-ins may use the grammars in the grammar database to recognize semantic categories. In a preferred embodiment, only certain grammars are designated for use for semantic category recognition and the list of the designated grammars would be obtained from the schema registry manager. A more detailed description of using grammars to recognize semantic categories is provided below.

Referring now to Fig. 7, a flowchart illustrating a method **700** for semantic category recognition using grammars in accordance with an embodiment of the present invention will be described. At step **705**, the user starts an application that in turn boots several recognizer plug-ins. At step **710**, one or more of the recognizer plug-ins communicates with the schema registry manager and retrieves a complete set of schema/grammar pairs to use for semantic category recognition. At step **715**, the user enters some text in a place in a document where there is no mode bias. At step **720**, during idle time, the text is passed to the recognizer plug-ins. At step **725**, the recognizer plug-ins apply the schema from the schema registry to the text if the text conforms to the grammars found in the schema registry. At step **730**, when the document is saved, the text is saved along with the schema applied to that text. Thus, as will be understood from the foregoing description, the schema has been used to label the text as a semantic category. The action plug-ins may then use this schema to provide actions.

Thus, as described above, applying mode bias in accordance with the present invention may also be related to recognizing semantic categories. Recognizing semantic categories requires determining what category some text is in after it is entered whereas applying mode bias requires specifying in advance what type of text will be entered. The same rules or statistical models (grammars) can be used to recognize semantic categories by determining if the text matches any of the rules and for specifying mode bias by constraining input with a probabilistic input method to conform to those rules.

Thus, in one embodiment of the present invention, once the user has entered a phone number into a field that has phone number mode bias set, the fact that that field contains a phone number is retained. Thus, the fact that the field is a phone number will be apparent to whatever person (such as a user who requests a list of actions pertinent to that data type) or process (e.g. a search engine) looks at the document later. Moreover, if the user had entered a phone number in an arbitrary location in a document—that is, in a place where there was no a priori specification of mode bias—the phone number will be labeled as such by recognizer plug-ins. That is, once the mode bias mechanism of the present invention is in place, a recognizer plug-in may identify many of the items in the schema registry using the same patterns that are used for input. However, not all of the grammars necessarily have to be used because some grammars may be for particular genres of text, such as legal text or medical text, or even something as specific as case law text. There may be little use in labeling all text that appears to be from one particular genre. Similarly, some grammars are so generic as to be useless. For example, generic types such as numbers may not facilitate good semantic category actions.

Thus, it should be understood from the foregoing description that the same grammar could be used to apply mode bias and also to recognize semantic categories. Leveraging this relationship is advantageous for developers who will have fewer grammars to build and for users who will receive better input and richer functionality via semantic categories. In addition, the relationship between mode bias and semantic categories maximizes consistency because if something is labeled as a part number semantic category then you should be able to enter it when a form requests a part number and vice versa.

It should be understood from the foregoing description that it is important to have a standard way to specify that input into a particular region of a document or form should conform to some pattern because the expected input is of a particular type or has particular properties. The present invention provides a way to specify that input into a particular region of a document or form should conform to some pattern. For example, when entering a telephone number into a telephone number field of contact management

software, the user will almost certainly enter a well-formed phone number and whatever input method they're using should make that as easy as possible. The exact form of the phone number may differ depending on the user's location, but wherever the user is in the world, it's unlikely that they'll expect to be able to enter free text into a telephone number field.

It should be understood that tools other than input methods, such as speech, handwriting or East Asian input method editors, also need to know about mode bias. For example, proofing tools such as spelling, grammar and style checkers should be able to adapt to the mode bias of the present invention to some extent. For example, if text is entered into a form that has mode bias using a non-probabilistic input method (such as typing English), then whatever proofing tools are run on that text should be cognizant of the constraints placed on that text by the mode bias. For example, it may make sense to turn spell checking off in fields that allow names to be entered. Or, a speller checker that uses a statistical language model might be able to swap in a more appropriate model with the knowledge that the text is medical or legal in nature.

It should be understood from the foregoing description that the present invention allows all input methods to become standardized in their handling of mode bias. For example, for a telephone number input, a speech recognizer and a handwriting recognizer will receive the same (or a similar) grammar from the schema registry manager. Then, the speech recognizer and handwriting recognizer will be able to handle telephone number inputs similarly and there will be a standardization of input methods. The present invention is also more flexible than the prior art because new schemas and grammars may be added to the databases and may be updated as necessary.

It should be understood from the foregoing description that after an input method receives a grammar, it may use it in many different ways that will be apparent to those skilled in the art. For example, in the case of a context free grammar, an input method may attempt to match what the user said (or typed or handwrote, etc.) with elements of the grammar and, if not present, then reject the input. As another example, the input method may use the grammar as a way to disambiguate words, phrases, etc. (e.g. if the

grammar includes the word "two" but not "to", this information may be used to recognize what the user is inputting).

It should be understood that the foregoing pertains only to the preferred embodiments of the present invention, and that numerous changes may be made to the embodiments described herein without departing from the spirit and scope of the invention.